remote sensing

MDPI

Technical Note

Loss of Significance and Its Effect on Point Normal Orientation and Cloud Registration

Matthew Young [1,*], Chris Pretty [1], Sérgio Agostinho [2], Richard Green [3] and Xiaoqi Chen [1]

[1] Department of Mechanical Engineering, University of Canterbury, Christchurch 8041, New Zealand; chris.pretty@canterbury.ac.nz (C.P.); xiaoqi.chen@canterbury.ac.nz (X.C.)

[2] Institute for Systems and Robotics (ISR/IST), Instituto Superior Técnico, Universidade de Lisboa, 1049-001 Lisboa, Portugal; sergio.agostinho@tecnico.ulisboa.pt

[3] Computer Science and Software Engineering Department, University of Canterbury, Christchurch 8041, New Zealand; richard.green@canterbury.ac.nz

* Correspondence: msy22@uclive.ac.nz

Received: 22 April 2019; Accepted: 28 May 2019; Published: 3 June 2019

Abstract: Point normal calculation and cloud registration are two of the most common operations in point cloud processing. However, both are vulnerable to issues of numerical precision and loss of significance. This paper documents how loss of significance in the open-source Point Cloud Library can create erroneous point normals and cause cloud registration to fail. Several test clouds are used to demonstrate how the loss of significance is caused by tight point spacing and clouds being shifted far from the origin of their coordinate system. The results show that extreme loss of significance can cause point normals to be calculated with a random orientation, and cause meters of error during cloud registration. Depending on the structure of the point cloud, loss of significance can occur when the cloud is at hundreds or even tens of meters from the origin of its coordinate system. Shifting to larger data types (e.g., from 32-bit "floats" to 64-bit "doubles") can alleviate the problem but will not solve it completely. Several "best practice" recommendations for avoiding this issue are proposed. But the only solution guaranteed to eliminate loss of significance is de-meaning the entire cloud, or clusters of points before processing.

Keywords: point normal; point clouds; numerical precision; LiDAR; Point Cloud Library; Iterative Closest Point

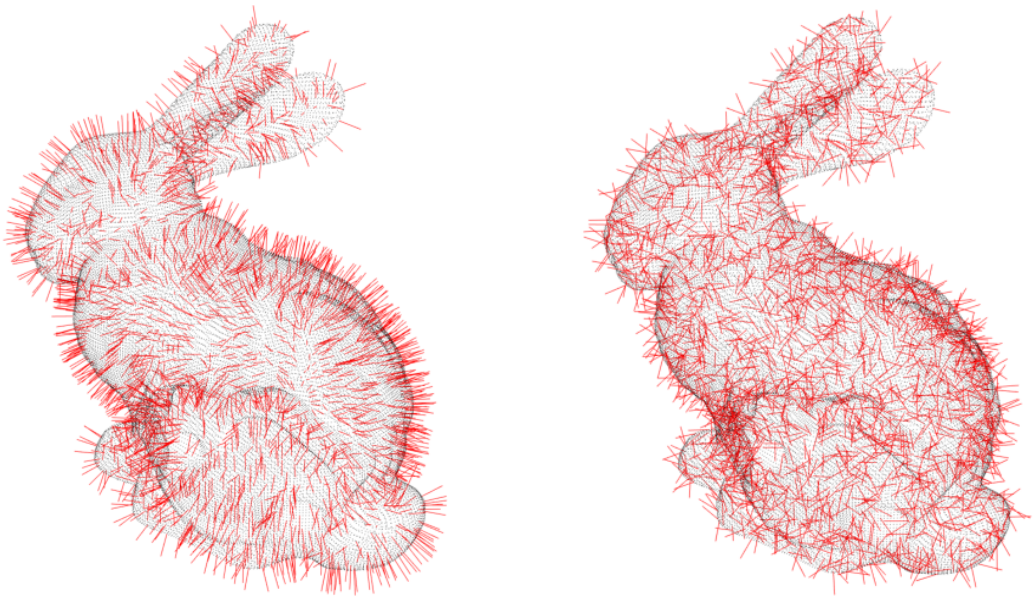
1. Introduction

Much research has been done to date on the development of point cloud registration and normal calculation algorithms. However, relatively little research has been done to ensure that the way they are implemented is robust and that their results are reliable.

While many algorithms for point cloud processing have infinite precision in theory, in practice they often require making compromises between precision and computation speed. This can introduce problems with loss of significance (LoS), where errors can be introduced that may have consequences for every down-stream process. This is of particular significance for normal calculation since it is often a precursor to other cloud processing operations such as feature identification or surface reconstruction. Specifically, LoS occurs when the result of an arithmetic operation is too big to be accurately stored in the container type assigned to it. The result is then rounded or clipped to a number that the container can represent. Unfortunately, the problem does not end there as this erroneous number may be involved in further arithmetic, propagating the error through an entire function and adding a considerable amount of error to the output.

In the context of cloud registration, this problem will manifest as a poor cloud match. In the context of normal vector calculation, the error may be obvious such as a "NaN" (Not-a-Number) or

"Inf" (Infinite number, typically the result of a division by zero). Alternatively, the normal vector may be calculated with an angular deviation from its true value. This is easy to observe with a simple point cloud, such as the Stanford Bunny. When the cloud is at origin, the normals (represented as red lines) intuitively all point outwards from the center of the cloud. However, when the cloud is pushed an arbitrarily large distance from the origin, as shown in Figure 1b, the normal vectors now appear to have random orientations.



(**a**) Correct normal orientation (**b**) Incorrect normal orientation caused by loss of significance

Figure 1. Demonstration of how loss of significance occurs when a cloud is pushed far from the origin of its coordinate system, and the effect this has on point normal orientation.

The open-source Point Cloud Library (PCL) [1] is an extensive library for processing point cloud data. It is widely used in academia for a variety of computer vision, mapping and robotic applications. Issues of numerical precision and LoS are a concern for all point cloud processing software, and PCL is no exception. Because of its open-source and easily accessible nature, PCL is used by students and others who are relatively inexperienced with these issues, and thus are often unaware of the potential pitfalls and care that needs to be taken in some point cloud applications, particularly applications involving LiDAR and large real-world datasets, where the magnitude of the point coordinates makes the clouds susceptible to LoS.

Further compounding the problem is the fact that it is not common practice in this field for users to visualize and explicitly check their point normal orientations to ensure they are accurate. As for registration, if a sub-optimal cloud match is obtained, it is often assumed to be the result of poor tuning or cloud quality, not a subtle failure on the part of the implemented algorithm. With this in mind, the contribution of this paper is as follows: it explains how LoS can occur in normal vector calculation and cloud registration. It demonstrates this phenomenon with PCL, and then proposes several best-practice recommendations for mitigating potential LoS. It should be noted that, while this paper refers exclusively to PCL, and describes what should be considered an extreme case; LoS will affect all cloud processing software to a greater or lesser degree.

2. Background

2.1. Normal Vector Calculation

To maintain consistency with existing literature, this paper employs the same notation adopted by Klasing et al. [2] and Jordan et al. [3], who denoted a cloud of N points as $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_N\}, \boldsymbol{p}_i \in \mathbb{R}^3$, each point as a triplet $\boldsymbol{p}_i = [p_{ix}, p_{iy}, p_{iz}]^T$ with an associated normal vector $\boldsymbol{n}_i = [n_{ix}, n_{iy}, n_{iz}]^T$. Any given point $\boldsymbol{p}_i$ for which we want to calculate the normal of has a neighborhood of k points $Q_i = \{\boldsymbol{q}_{i1}, \boldsymbol{q}_{i2}, ..., \boldsymbol{q}_{ik}\}, \boldsymbol{q}_{ij} \in P, \boldsymbol{q}_{ij} \neq \boldsymbol{p}_i$.

The normal vector $\boldsymbol{n}_i$ for any given point $\boldsymbol{p}_i$ in a cloud is determined by its surrounding neighborhood of points Q_i, which together with $\boldsymbol{p}_i$ typically approximate some shape or laser-scanned surface. In some applications, the cloud is given as a square or triangular mesh, in which case the point normals are referred to as vertex normals, and can be calculated directly from the mesh using neighboring triangles or vertex angles. [4]. However, in many cases involving real-world data such as that produced by a LiDAR unit, this mesh is not available and the point normals must be calculated from the structure of the cloud itself.

Most normal calculation methods operate by fitting a plane [5], surface or other geometric model [6] to $\boldsymbol{p}_i$ and its neighbors Q_i. Errors between points and the model are minimized using processes such as Singular Value Decomposition (SVD) or Principal Component Analysis (PCA), and then it is from the model that the normal vector is derived. A good explanation and comparison of such methods can be found in the works of Klasing et al. [2] and Jordan et al. [3]. Some methods may be modified by applying weights [5,7] or bounds [8] to improve the accuracy of the solution.

While these studies often discuss the accuracy and computational speed, they do not discuss the stability of these methods, or how they can be affected by numerical precision and the ensuing consequences. Thus, at the time of writing, there appears to be no research which directly addresses numerical precision and its effect on point cloud operations such as point normal calculation and cloud registration.

2.2. Calculation of Normals with Eigenvalues

The most widely used method to calculate the normal for a point is to use a Least-Squares formulation to fit a plane to the point $\boldsymbol{p}_i$ and its k-nearest neighbors. The normal is the set of values n_{ix}, n_{iy}, and n_{iz} that best satisfy the equation for a plane, given by Equation (1).

$$a\, n_{ix} + bn_{iy} + cn_{iz} + d = 0 \tag{1}$$

This process requires calculating a three-dimensional co-variance matrix for the cluster of points, the standard form of which is given by Equation (2), with variance on the diagonal and co-variance on the off-diagonal.

$$\mathbf{C} = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(x,y) & cov(y,y) & cov(y,z) \\ cov(x,z) & cov(y,z) & cov(z,z) \end{bmatrix} \tag{2}$$

This matrix encodes the variance of the points in 3D space, where each eigenvector lies along a principal component of the cluster and the corresponding eigenvalue represents the magnitude of the variance in that direction. If the points are structured and approximate a plane, their coordinates will vary along the length and width of the plane significantly more than in the direction of its depth or "thickness". Therefore, the smallest eigenvalue of $\mathbf{C}$ corresponds to the eigenvector that lies orthogonal to this plane, in the direction of least variance. The normalized eigenvector is the point normal.

The co-variance of two discrete variables (in this case any combination of p_{ix}, p_{iy} or p_{iz}) is defined as the mean product of the difference between each variable and its mean, as shown in Equation (3). This can be expanded out using the linearity property of expectations.

$$\begin{aligned} cov(x,y) &= E[(x - E[x])(y - E[y])] \\ &= E[xy] - E[x]E[y] \end{aligned} \tag{3}$$

A simplified 2D version of the expanded form of Equation (3), as implemented by PCL, is given in Algorithm 1. In practice, the functional code calculates the co-variance of each combination of coordinates ($cov(x,x)$, $cov(x,y)$, $cov(x,z)$, etc.) and the co-variance matrix itself, all within the same function. The specific section of PCL code that corresponds to Algorithm 1 is the `computeMeanAndCovarianceMatrix` function, defined in the source file `centroid.hpp`.

Algorithm 1 Co-variance calculation.

Input: A cloud of N points
Output: Co-variance of x and y for the cloud
1: initialize array *mean*
2: **for** *point* **in** *cloud* **do**
3: $mean[0] = mean[0] + point.x$
4: $mean[1] = mean[1] + point.y$
5: $mean[2] = mean[2] + point.x * point.y$
6: **end for**
7: $mean = mean/size\_of(cloud)$
8: $cov = mean[2] - mean[0] * mean[1]$
9: **return** *cov*

This method of computing the normal vector was first proposed by Hoppe et al. [5] in 1992 and this method, or a close variation of it, has been used in numerous studies since [2,6,8–10].

2.3. Point Cloud Registration with Iterative Closest Point

As with normal vector calculation, a wide variety of algorithms exist for performing point cloud registration, but there are a few key algorithms that are used by most researchers in this field. In the case of cloud registration, this is the family of algorithms known as Iterative Closest Point (ICP).

First introduced by Besl and McKay [11] in 1992, the goal of ICP is to match or "register" two point clouds. In the simplest form of this problem, two clouds A and B have m points in the sets $A = \{\boldsymbol{p}_{1a}, ..., \boldsymbol{p}_{ma}\}$ and $B = \{\boldsymbol{p}_{1b}, ..., \boldsymbol{p}_{mb}\}$, respectively. $\boldsymbol{p}_{ia}$ and $\boldsymbol{p}_{ib}$ are corresponding points. The standard "Point-to-Point" ICP algorithm then finds a transform T that aligns the clouds by minimizing an error function, as shown in Equation (4).

$$E_{ICP}(\boldsymbol{T}) = \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{T}\boldsymbol{p}_{ia} - \boldsymbol{p}_{ib}\|^2 \tag{4}$$

ICP algorithm starts with an initial guess for each match and then iteratively improves the estimate of $\boldsymbol{T}$ by re-matching pairs and re-computing the error function, eventually converging to a minimum.

In practice, this problem is often more complicated. The clouds may be non-identical, may have a poor initial alignment, may sample the same object at different points ($\boldsymbol{p}_{ia} \neq \boldsymbol{p}_{ib}$), may only partially overlap, may contain many sources of error and so on. Under ideal conditions, the algorithm will converge to a global minimum, but it is well known that ICP can fall into a local minimum instead.

An alternative version of the ICP algorithm models the surface between neighboring points in cloud B and then minimizes the distance between that surface and the point cloud A. This is referred to as "Point-to-Plane" ICP (PICP) [12], and modifies the ICP error function as shown:

$$E_{PICP}(\boldsymbol{T}) = \frac{1}{N}\sum_{i=1}^{N}((\boldsymbol{T}\boldsymbol{p}_{ia} - \boldsymbol{p}_{ib})\boldsymbol{n}_{iab})^2 \tag{5}$$

where $\boldsymbol{n}_{iab}$ is orthogonal to the modeled surface in B and runs between the surface and point $\boldsymbol{p}_{ia}$. This algorithm intelligently minimizes the error along the direction of the surface normals, while allowing the clouds to slide in directions orthogonal to the normals.

The latest version of the algorithm extends this further by using co-variance matrices of point neighborhoods to model and align the cloud surfaces directly:

$$E_{GICP}(\boldsymbol{T}) = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{d}_i^T(\boldsymbol{C}_i^B + \boldsymbol{T}\boldsymbol{C}_i^A\boldsymbol{T}^T)^{-1}\boldsymbol{d}_i \tag{6}$$

where $\boldsymbol{C}_i^B$ and $\boldsymbol{C}_i^A$ are the co-variance matrices of corresponding points $\boldsymbol{p}_{ia}$ and $\boldsymbol{p}_{ib}$, and $\boldsymbol{d}_i = \boldsymbol{p}_{ib} - \boldsymbol{T}\boldsymbol{p}_{ia}$ is a vector of their point-to-point distances. Note that here $\boldsymbol{d}_i^T$ and $\boldsymbol{T}^T$ are transposes. This is called "Plane-to-Plane" or "Generalized" ICP (GICP) [13]. By using co-variance matrices of point neighborhoods directly, GICP does not calculate or require normal vectors. Many other variations of the standard ICP algorithm exist, however standard ICP and PICP are currently the most widely used within the research community.

PCL includes implementations of all three described ICP variants, in addition to a few other registration algorithms such as the Normal Distribution Transform (NDT) [14]. Of the ICP variants, GICP is the most accurate [13].

For a comprehensive breakdown of the ICP process, readers should refer to the work of Rusinkiewicz and Leroy [15]. For further reading on the practical application of ICP, the works of Pomerleau et al. [16] and Donoso et al. [17] are good starting points.

2.4. Single Precision and Loss of Significance

In mathematics, the decimal precision of any given value can be arbitrarily large. However, in software, a value must be stored in a variable of a finite size. For reasons of computational efficiency, most point cloud processing is done in the C or C++ languages with 32-bit, single-precision, floating point numbers as defined by the IEEE 754 standard [18]. That is, each point coordinate (p_{ix}, p_{iy} or p_{iz}) is stored as an individual "float" which can accurately represent seven or fewer significant digits [19]. Some functions in PCL and other point cloud processing libraries such as Libpointmatcher give the user the option to use 64-bit, double-precision numbers ("doubles") [16]. Although as this paper will show, increasing the size of the variable data type simply increases the threshold at which LoS will occur.

Computer code such as that shown in Algorithm 1 is particularly susceptible to LoS not only because it is summing many (potentially large) point coordinates but because it is summing the result of their multiplication. A full numerical analysis of the relevant parts of the PCL code base is well beyond the scope of this paper, but a small example with two points is useful in illustrating the effect. A reader interested in learning more about numerical precision should refer to the work by Goldberg [20].

Let $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ be two points whose x and y coordinates require exactly seven significant digits to be accurately stored to the nearest millimeter:

$$\begin{array}{ll} p_{1x} = 6273.544 & p_{1y} = 5180.157 \\ p_{2x} = 6273.794 & p_{2y} = 5180.657 \end{array}$$

If these points are input to Algorithm 1, we see that some arithmetic operations produce an incorrect result, as shown by Table 1 where the incorrect digits are underlined.

In this example, the errors that occur are on the order of centimeters. However, it is clear that the more points there are in the cloud or the further it is from origin, the greater the LoS will be. As shown below, the problem is also a function of point spacing. Because the smaller the spacing is between points, the more severe the error is when significant digits are lost.

When Algorithm 1 is used in practice, the size of the cloud is determined by the number of nearest neighbors k required to calculate the normal. Since this number is usually important to the accuracy of the calculated normal, it can be treated as a constant, in which case the LoS is *effectively* a function of just the point distance and spacing.

LoS also occurs at various stages of each ICP variant, as their implementations also require a significant number and variety of arithmetic operations. Their implementation in PCL is significantly more complex than that of point normal calculation. Thus, while the next section explains how LoS occurs during normal calculation in detail, a similar analysis of the ICP variants is beyond the scope of this paper.

Table 1. Calculated vs. correct results of Algorithm 1 with two example points.

Line	Operation	Correct Result	Single Precision
3	$\sum p_{ix}$	12547.338	12547.33$\underline{7}$
4	$\sum p_{iy}$	10360.814	10360.814
5	$\sum p_{ix}p_{iy}$	65000320.000	6500031$\underline{7.669}$
7	$0.5\sum p_{ix}$	6273.669	6273.669
	$0.5\sum p_{iy}$	5180.407	5180.407
	$0.5\sum p_{ix}p_{iy}$	32500160.000	3250015$\underline{8.825}$
8	See Equation (3)	−3612.209	−3612.$\underline{000}$

2.5. Loss of Significance in PCL Calculation of Normal Vectors

The point normal calculation process in PCL is illustrated in Figure 2, as it is implemented at the time of writing. This process and individual function names may change after publication, however this figure has been included because it is invaluable in illustrating the problem.

When LoS occurs, it can change the value of the terms in Equation (2). This means that the result can be wrong. However, it also means that, when computing the variances along the diagonal, the $E[xx]$ (or $E[yy]$ or $E[zz]$) term can be rounded down and become smaller than the $E[x]E[x]$ term, resulting in a negative variance which is theoretically impossible. The characteristic equation for $\mathbf{C}$ is sensitive to error in every element of the matrix. So when LoS occurs and an element is calculated incorrectly the resulting eigen values and vectors of $\mathbf{C}$ have a highly random nature.

As stated above, the problem is a function of cloud distance from origin and point spacing, and that it has a negative effect on cloud registration. Empirically demonstrating this effect is the focus of the remainder of this paper.

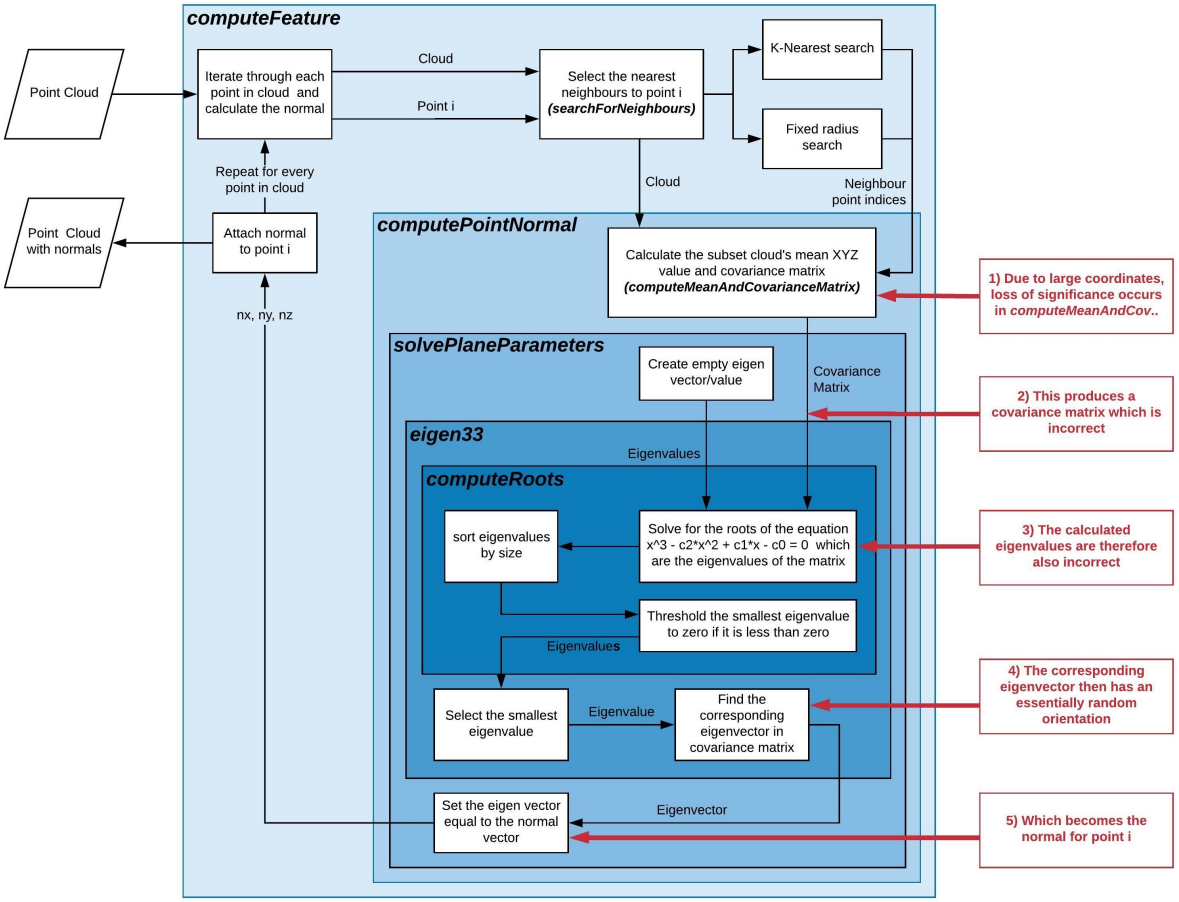


Figure 2. Flow diagram outlining the PCL process for calculating point normals and how LoS causes them to be calculated inaccurately. The key stages of the numerical precision issue are shown in the red text boxes. Names of the relevant functions are shown in bold italics.

3. Experimental Setup

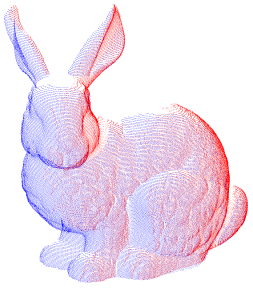
3.1. Choice of Test Datasets

When choosing point cloud datasets with which to conduct empirical experiments, some researchers use publicly available data, their own generated clouds [2,5], mathematically defined clouds [4] or a combination of the three [3,9].

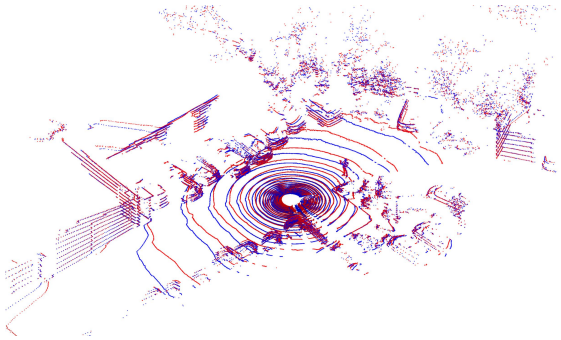
For this study, a selection of three fabricated and real-world point clouds were chosen that reflect some of the common applications PCL is used for. Namely, the well-known Stanford Bunny to reflect the common use of PCL in manipulating small object point clouds, a LiDAR scan from a Velodyne HDL-32 to represent PCL's use in SLAM, and a Trimble SX10 scan to reflect the uses of PCL in geoscience and mapping. These clouds are shown in Figure 3.

All three sets consist of an A and B cloud, which overlap but are non-identical. The Stanford Bunny dataset is composed of two of the original scans from the Stanford 3D Scanning Repository, specifically scans 45 and 315, which have a size of approximately 40,000 and 35,000 points, respectively. They have not been scaled in any way, and so retain their original dimensions of approximately 0.15 m $\times$ 0.15 m $\times$ 0.12 m.

The Velodyne HDL-32 carpark scans have dimensions of 43 m $\times$ 26 m $\times$ 5 m and a size of approximately 56,000 points each. The Trimble SX10 clock tower clouds have dimensions of 49 m $\times$ 40 m $\times$ 10 m and a combined size of approximately 4.5 million points.



(**a**) Dataset 1: Scans 45 and 315 of the Stanford Bunny



(**b**) Dataset 2: Two Velodyne HDL-32 LiDAR scans of a carpark

(**c**) Dataset 3: Two Trimble SX10 LiDAR scans of a clock tower

Figure 3. Test datasets. Each dataset has been chosen to represent a type of point cloud and application that PCL is commonly used for. Each dataset consists of an A (red) and B (blue) cloud which overlap but are non-identical.

Each pair of scans has already been correctly positioned relative to one another. Either by registering them with another application (Stanford Bunny), GPS positioning (HDL-32 scan) or total station positioning (SX10 scan).

It should be noted that these clouds are expressed in units of meters, as this is the native unit that the HDL-32 and SX10 datasets are created in. However, in the context of numerical precision, the units are arbitrary as the number of significant digits required to accurately store a number is the same regardless.

It is also worth noting that basic point clouds with mathematically definable normals (e.g., sphere, cube, and pyramid) were deliberately not selected. This is because the point normal calculation methods described and tested in this paper would inherently produce slightly different normals, particularly near sharp edges of the object. Thus, any results would show some amount of non-zero error that was not induced by LoS.

3.2. Metrics

Normal Orientation Error: If the ground truth normal vector of a point is $\boldsymbol{n}_i$, then the angular error in radians between it and the calculated normal vector $\hat{\boldsymbol{n}}_i$ is given by Equation (7). Note that both vectors must be normalized.

$$\theta_i = arccos(\hat{\boldsymbol{n}}_i^T \boldsymbol{n}_i) \tag{7}$$

Mean Point Spacing: Mean point spacing $\bar{s}_i$ is defined as the mean Euclidean distance between any point $\boldsymbol{p}_i$ and its nearest k neighbors in the set Q_i, formally given by Equation (8).

$$\bar{s}_i = \frac{1}{k}\sum_{j=1}^{k}\|\boldsymbol{p}_i - \boldsymbol{q}_{ij}\|_2 \tag{8}$$

Each of these metrics can be calculated per-point, where i denotes the point index, or they can be averaged over the whole cloud to produce a per-cloud mean. Note that calculating the mean point spacing for a whole cloud is only valid if the points are approximately uniformly distributed over the surface of the object, i.e., in the case of the Stanford Bunny clouds.

Distance-Density Ratio: This paper also introduces the concept of a distance–density ratio, which is simply the Euclidean distance of the point from origin, divided by the mean point spacing.

$$DDR_i = \frac{\|\boldsymbol{p}_i\|_2}{\bar{s}_i} \tag{9}$$

As shown Section 4.2, the larger the ratio is the more likely it is that LoS will occur and the more severe it will be.

Registration Error: Each dataset consists of two overlapping, non-identical clouds that are already correctly positioned relative to one another. When testing a cloud registration algorithm, a copy of cloud B is made and translated a known distance from its original position, this is the initial error. The error-added copy B′ is then registered to cloud A using one of the ICP variants described in Section 2.3. If the registration is perfectly accurate, cloud B′ should exactly match cloud B. If not, there will be some non-zero error that can be described as the mean Euclidean point-to-point distance between each point $\boldsymbol{p}_i$ in cloud B and the same point B′:

$$\bar{\varepsilon}_r = \frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{p}_{ib} - \boldsymbol{p}_{ib'}\|_2 \tag{10}$$

As with all other distances, registration error is expressed in meters.

3.3. Source of Ground Truth Normal Orientation

The "true" orientation of any normal vector can only be known if it can be determined mathematically (as with a sphere, cube or plane). Virtually all normal calculation algorithms rely on the surrounding points to define the curvature of the shape, and therefore rely on the point accuracy and density. Error, either added deliberately or acquired from sensor noise, makes determining the true normal impossible in most circumstances. Thus, what is calculated instead is simply the "best achievable" estimate of the "true" or "ground-truth" point normal. This is the case for all datasets shown in Figure 3.

In the context of *this* paper, the ground truth for each point normal is still calculated using the PCL process described in Section 2.5 and illustrated in Figure 2. However, for each "true" normal $\boldsymbol{n}_i$ to be calculated, the relevant point $\boldsymbol{p}_i$ and its nearest neighbors Q_i are de-meaned by subtracting the coordinates of $\boldsymbol{p}_i$. This process centers each cluster of points on the origin and sets the DDR to zero, thereby minimizing any possible LoS. This is the sole method used for computing the ground-truth normals for every cloud in every test. To reiterate, this is the "ground-truth" data in the sense this is the best achievable estimate of the normal, and is assumed to be error-free relative to the same calculations performed without de-meaning.

3.4. Hardware and Software Specifications

All experiments were conducted on a Dell Precision M3800 laptop with an Intel Core i7-4712HQ 2.30 GHz processor running an Ubuntu 14.04 operating system. Each experiment was written in C++11 and compiled with CMake version 3.12.1 using GCC and G++ versions 4.8.4. The version of PCL used was 1.8.1, which at the time of writing was the latest available version of PCL.

4. Effect of Loss of Significance on Normal Orientation Error

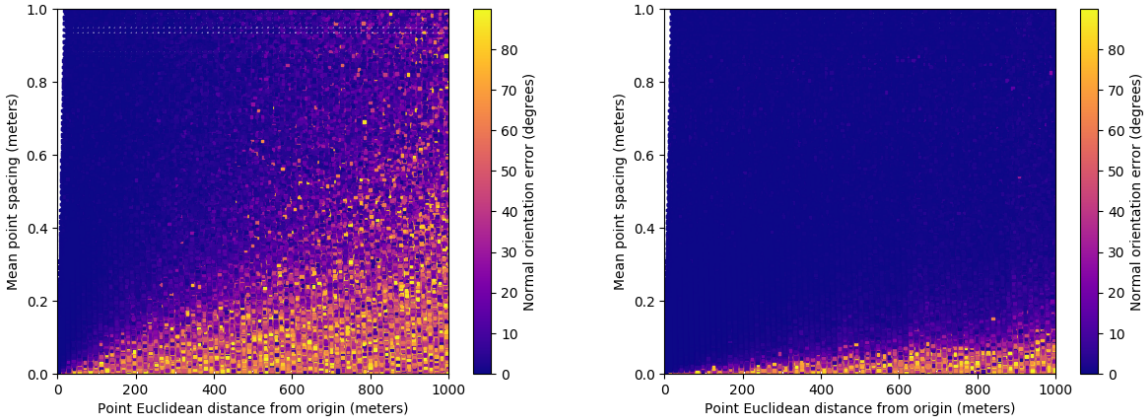
4.1. Normal Error Methodology

The normal orientation error is a function of cloud distance from origin and mean point spacing. Thus, to quantify their effect, the "B" cloud from each of the test datasets was placed at origin and then progressively increased in scale (to increase the mean point spacing) and moved further from the origin. When translating the cloud away from the origin, the whole cloud was shifted equally along all three axes to increase the magnitude of the point coordinates. Note that this methodology deliberately only uses the "B" cloud, ignoring the "A" cloud.

Prior to this, a number of M randomly chosen points were picked from each cloud. At each new scale and distance, each of the selected points had their mean point spacing ($\bar{s}_i$), Euclidean distance from origin ($\|\boldsymbol{p}_i\|_2$) and normal orientation error (θ_i) calculated as described in Section 3.2. In choosing a value for the sample size M, 10% of the cloud size was used. This was done to reduce computation times.

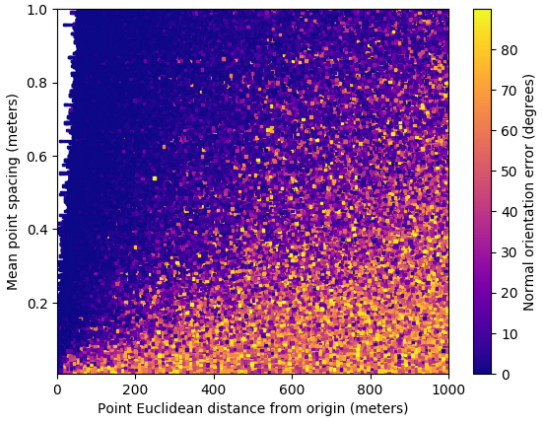
A k-nearest neighbor search was used to identify the set of neighbors Q_i closest to each point. In the choice of k, a value of 15 was selected as this is a value commonly used by existing studies [2,8]. As long as k is within an appropriate range, its exact value is not critical as research has shown that result of the LSQ normal calculation method is consistent for many similar values of k [2,3].

This experiment was conducted once with the data type set to float, and again with it set to double. Both sets of results are shown in Figures 4 and 5, with "float" results shown in the left-hand sub-figures, and "double" results shown in the right-hand sub-figures.

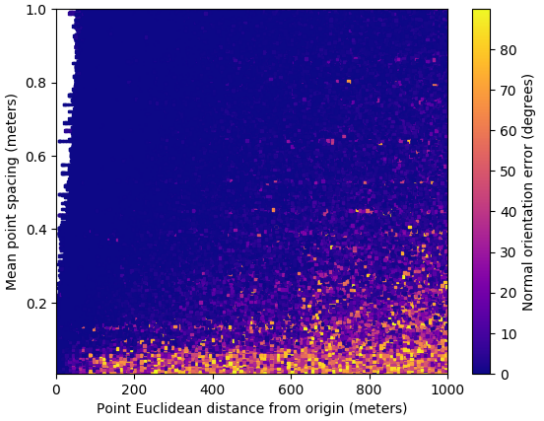


(**a**) Stanford Bunny (float) (**b**) Stanford Bunny (double)

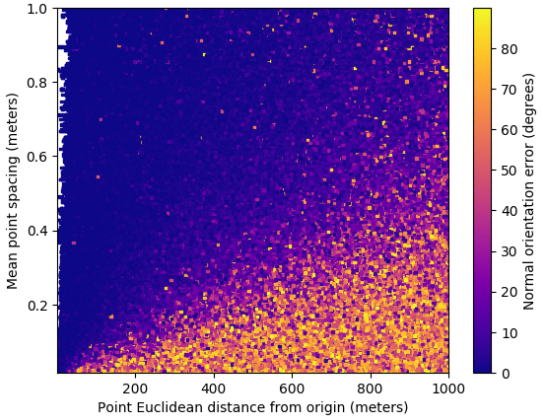
Figure 4. *Cont.*



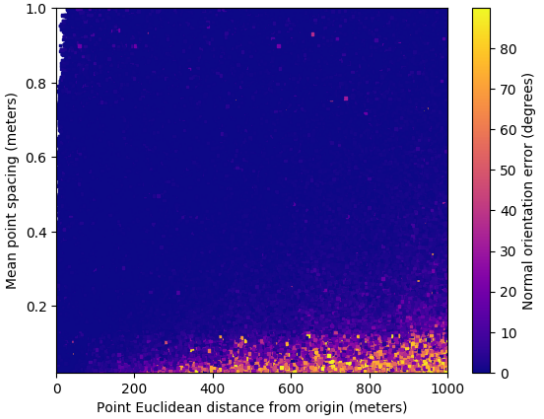
(**c**) HDL-32 carpark LiDAR scan (float)



(**d**) HDL-32 carpark LiDAR scan (double)

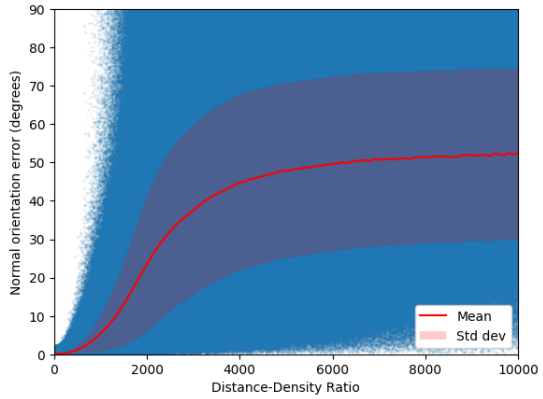


(**e**) SX10 clock tower LiDAR scan (float)

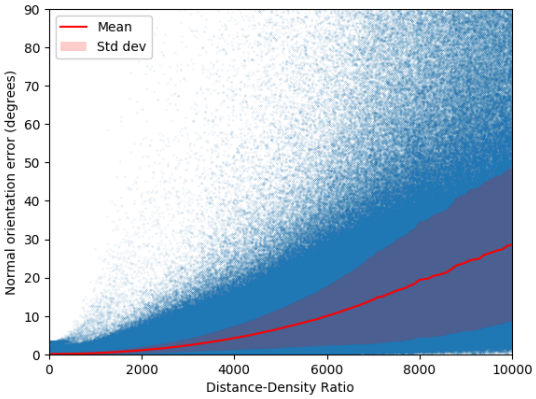


(**f**) SX10 clock tower LiDAR scan (double)

Figure 4. Normal deviation results. Scatter plots illustrate how loss of significance and normal error is a function of point spacing and point coordinate magnitude. Each plot point represents the calculated result of a single cloud point. Plots on the left were generated with the float data type, images on the right with the double data type.

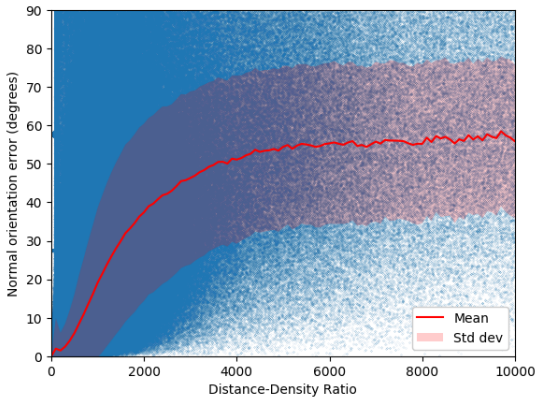


(**a**) Stanford Bunny (float)

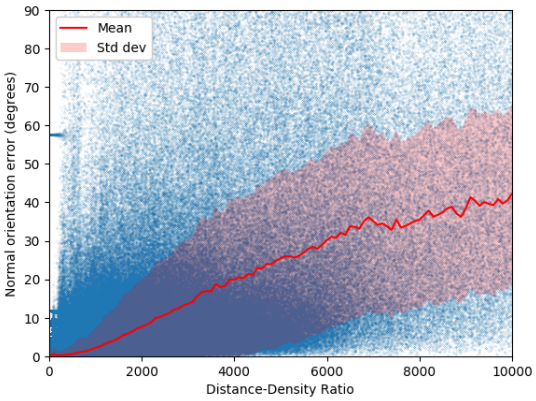


(**b**) Stanford Bunny (double)

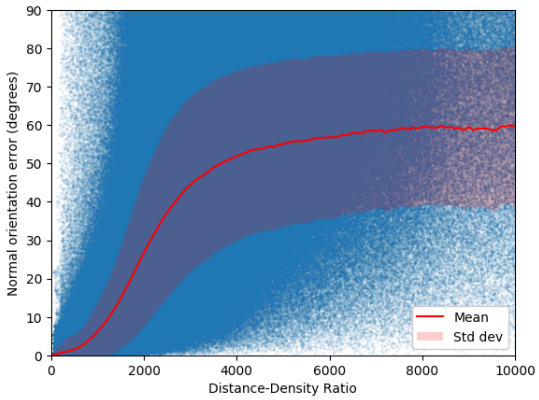
Figure 5. *Cont.*



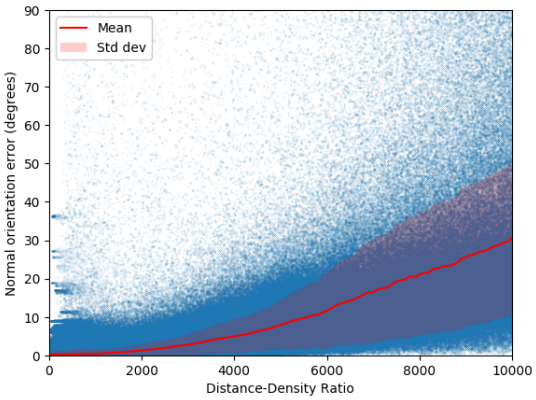
(**c**) HDL-32 carpark LiDAR scan (float)



(**d**) HDL-32 carpark LiDAR scan (double)



(**e**) SX10 clock tower LiDAR scan (float)



(**f**) SX10 clock tower LiDAR scan (double)

Figure 5. Normal error mean and standard deviation results. Scatter plots illustrate how as the DDR increases, so does the mean normal deviation from the ground truth. Each plot point represents the calculated result of a single cloud point. Plots on the left were generated with the float data type, images on the right with the double data type.

4.2. Normal Error Results

The results for normal testing on each of the three sample test clouds are shown in Figures 4 and 5.

The scatter plots in Figure 4 show that, when LoS does occur, the error is highly random, as indicated by the speckled pattern of colors which visually appears to cover the entire range of error from 0° to 90°. A uniformly low error only occurs when the point is close to origin, or has a large mean point spacing. The approximate DDR values at which the mean normal orientation error reaches a variety of thresholds are shown in Tables 2 and 3.

For all clouds the mean normal error increases as the DDR increases, which is clearly shown by the mean line in the scatter plots of Figure 5. In the left-hand plots, generated using the float data type, the mean orientation error increases rapidly before reaching a stable value of approximately 60° with a wide distribution, as shown by the standard deviation envelope. This result is consistent with the seemingly random orientations shown in Figure 1b. This is also mathematically consistent because a set of random vectors uniformly distributed over a hemisphere will have a mean error of exactly 60° from the vector that is orthogonal to the base of the hemisphere.

The right-hand plots, generated using the double data type, show the same trend. However, because the number of available significant digits was greater, the degree of LoS was less severe at each distance. As a result, the mean error increases slowly, and does not plateau until a much high DDR, off the end of the plots.

Table 2. Distance-Density Ratio at which the mean normal orientation error passes X° when calculated with floats.

Cloud	1°	5°	10°
Stanford Bunny	420	930	1290
HDL-32 carpark scan	60	410	620
SX10 clock tower scan	320	860	1220

Table 3. Distance-Density Ratio at which the mean normal orientation error passes X° when calculated with doubles.

Cloud	1°	5°	10°
Stanford Bunny	1890	4320	6000
HDL-32 carpark scan	600	1530	2250
SX10 clock tower scan	1740	3940	5570

When comparing datasets, the results reflect the nature of the cloud structure. For the Stanford Bunny, the normal error increases slowly, where the DDR must pass 420 before seeing even 1° of error in normal orientation. This means that the distance from any point in the bunny must be 420 times the mean spacing between the point and its nearest neighbors before 1° of error will occur in normal calculation. Most point clouds such as the Stanford Bunny are independent models or part of a scene, such as a depth image from a stereo camera. Since such clouds are typically centered at origin or processed in a small reference frame, it is unlikely that users in these types of applications will experience significant LoS. By contrast, Velodyne HDL-32 and Trimble SX10 are often used in geoscience and mapping applications, thus their point clouds are much larger in size with earth-referenced coordinates.

For the HDL-32 point cloud, the normal error is more pronounced and occurs much sooner, showing a mean of 1° at just 60 DDR. This is particularly concerning because the HDL-32 (and many similar LiDAR units) generate points with mean spacing on the order of a few centimeters or even millimeters. If the cloud is close to the origin, the normal error may be significant. In some extreme cases, if point density is very tight, and on the order of fractions of a millimeter, normal orientation error may occur within a cloud even if its centroid is at origin.

The particular vulnerability of the HDL-32 scan to normal error is partly because of the unique way in which such 3D spinning LiDAR units work. Because they consist of several single-beam LiDAR mounted along a spinning vertical axis, the point clouds they produce tend to bunch points together in bands, where each band corresponds to a different beam from the LiDAR. This irregular yet potentially tight spacing along the curve of the band is what is likely contributing to the randomized error seen in all parts of the scatter plot in Figure 4c,d.

The results of the SX10 clock tower scan are very similar to the results of the Stanford Bunny, which is intuitive as both clouds predominantly consist of flat or smooth surfaces. If the SX10 clock tower scans contained more organic objects such as trees or bushes, it is likely that the results would be closer to what is observed in the HDL-32 carpark scan results.

With every dataset, results calculated with the double data type clearly show that increasing the number of available significant digits reduces but does not eliminate the level of error caused by LoS. Using data types larger than 64-bit will have the same effect. The only guaranteed way to eliminate LoS is to de-mean the point and its nearest neighbors before computing the normal.

5. Effect of Loss of Significance on Cloud Registration

5.1. Registration Error Methodology

The final objective of this paper is to demonstrate the effect of LoS on cloud registration, which is ultimately what will concern most users of PCL. To do this, a similar process was used to the one outlined in Section 4.1. A few key differences were because while the DDR can be calculated on a per-point basis, it cannot be meaningfully calculated for a whole cloud, especially if that cloud has a large variation in point spacing. In addition, the registration error can only be expressed per-cloud.

To quantify the effect of LoS on cloud registration, each dataset was progressively shifted from origin, without altering its scale. Both clouds in each dataset were moved as one so that their relative position was maintained. At each increment from the origin, a copy of cloud B was made (B′) and translated by a fixed offset to simulate added error. The initial error was chosen to be 0.001 for the Stanford Bunny dataset, and 0.2 for the other datasets. Cloud B′ was then registered to cloud A using each described variant of the ICP algorithm. The PICP algorithm was used twice, once using normals calculated in-place, and again with the normals correctly calculated using de-meaning. The mean point-to-point distance of the registered cloud B′ and B was then calculated as the registration error, as described in Section 3.2.

Parameter tuning is an important part of cloud registration, as many registration algorithms (ICP variants included) may perform poorly without appropriate tuning. However, to re-iterate, one of the stated objectives of this paper is not to demonstrate how accurate each ICP method can be, but simply that their accuracy is affected by LoS. Thus, tuning each ICP variant for each dataset is beyond the scope of this paper. In addition, it would be unfeasible to provide tuned parameters for each ICP variant and dataset combination used in this paper. Thus, all ICP variants used their default parameters, as determined by the PCL code base. The only exception is that when registering the Stanford Bunny dataset, the maximum correspondence distance was set to 0.002.

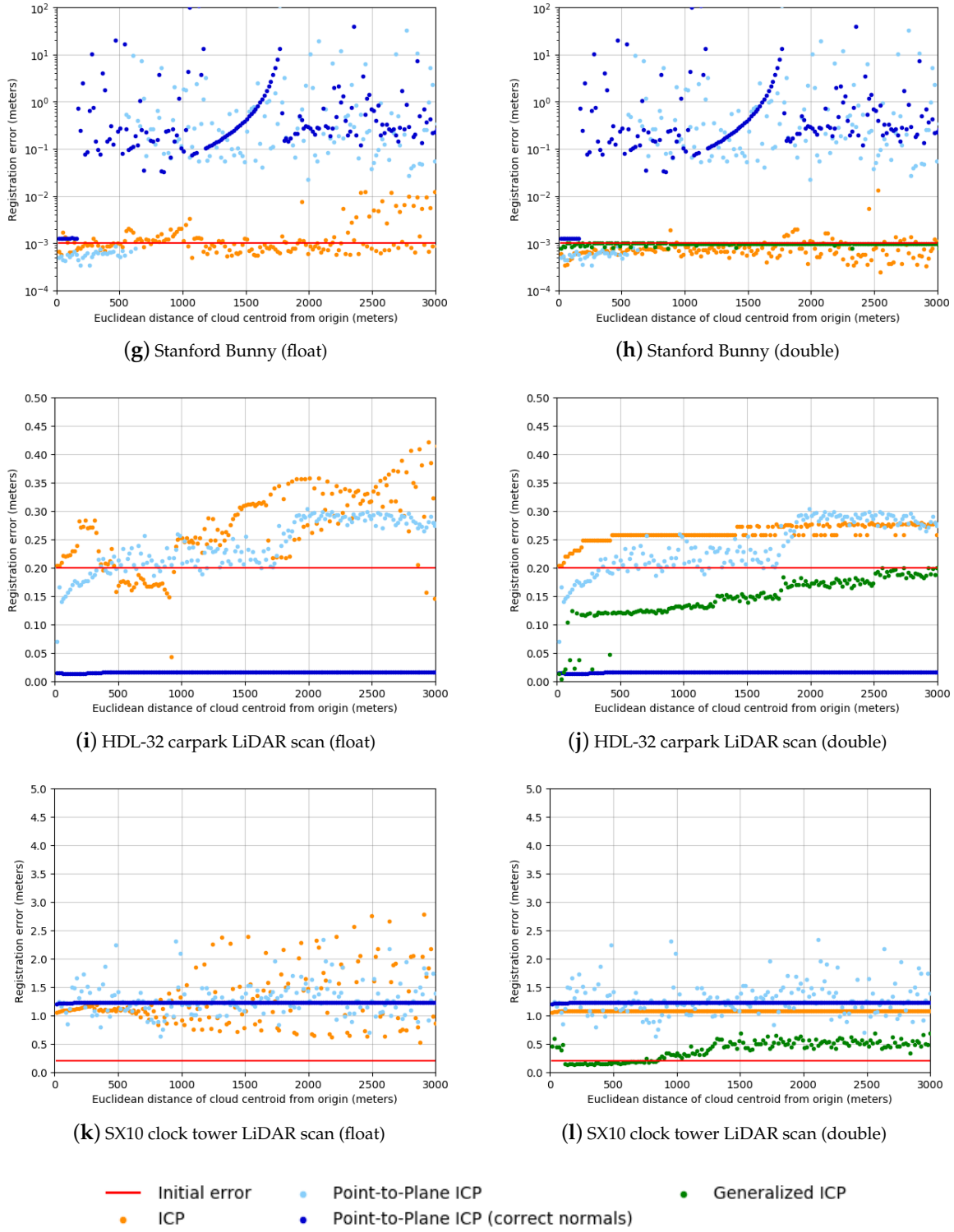
Like the normal orientation results, the registration results were calculated with both float and double data types. ICP and PICP use floats by default, however PCL's implementation of GICP is hard-coded to use doubles, and unlike the ICP and PICP implementations is not templated to allow the user to specify the data type. Thus, GICP results are only shown on the plots explicitly labelled "double".

In addition, note that PCL provides several implementations of point-to-plane ICP, this paper uses the Linear-Least Squares version as described by Low [21].

5.2. Registration Error Results

The results showing the effects of LoS on cloud registration are provided in Figure 6. While these results are subjective as they apply only to PCLs specific implementation of each ICP algorithm, they highlight how unpredictably the algorithms can behave when LoS starts occurring.

The standard ICP algorithm logically performs poorly with all three datasets. Because it simply tries to minimize point-to-point distance, it will never correctly match non-identical clouds. For each cloud pair, there will be some orientation that minimizes this distance yet incorrectly aligns them. When ICP is used to register all three datasets, it converges in an unstable manner, producing inconsistent errors that larger as LoS increases. When ICP is used with a double data type instead of float, its result is more consistent although still often worse than the initial error.



(**g**) Stanford Bunny (float)

(**h**) Stanford Bunny (double)

(**i**) HDL-32 carpark LiDAR scan (float)

(**j**) HDL-32 carpark LiDAR scan (double)

(**k**) SX10 clock tower LiDAR scan (float)

(**l**) SX10 clock tower LiDAR scan (double)

Figure 6. Registration point-to-point error. Each plot point represents the calculated result of a single cloud point. Plots on the left were generated with the float data type, images on the right with the double data type.

The registration results of the PICP algorithm show that its accuracy is solely determined by whether or not the point normals were calculated accurately. The standard PICP results of all three datasets are inaccurate and imprecise regardless of whether they are calculated using floats or doubles.

By comparison, when the PICP algorithm is used with correctly calculated normals, it provides precise results at every distance, and regardless of data type. This implies that correct normal calculation can make PICP impervious to LoS. However, during registration of the Stanford Bunny dataset, past approximately 200 meters the results become extremely inaccurate. This is because past 200 meters LoS causes correspondence detection to fail, and therefore the failure of the whole registration algorithm. The end result is the B′ cloud being transformed to a position far away from the A cloud.

In most cases, GICP produces results that are more accurate if not more precise than ICP or PICP. However, it is clear that GICP is still being affected by LoS, as evident by fact that the error increases with distance from origin. It has already been stated that PCL's implementation of GICP calculates the required co-variance matrices internally, and explicitly casts them to the double data type. However, there are still some components of the GICP implementation that are hard-coded to use floats, so it is possible that LoS is occurring elsewhere in the GICP process. This would explain why the GICP results appear to be less precise than PICP with correct normals. Unfortunately, confirming this hypothesis is outside the scope of this paper, but a deeper investigation of where exactly LoS occurs in different registration algorithms would make for valuable future research.

Regardless of the flaws of each ICP variant, all perform best when their clouds are very close to, or at the origin. As with the normal orientation results, increasing the size of the data type reduces the problem of LoS but does not eliminate it. Thus, the only way to eliminate the error caused by LoS is to de-mean both clouds prior to registration.

6. Best Practices for Avoiding Loss of Significance

Users of PCL as well as any other point cloud processing application will obviously want to avoid issues caused by loss of significance. This paper demonstrates its effects on normal orientation and cloud registration, however there are many other point cloud processes that can be affected.

The results of this paper prove that loss of significance can be avoided entirely by centering each point cloud at origin before operating on it. However, this is not always possible, especially in mapping and geoscience applications, where large Earth-referenced point clouds may be required. To this end, the authors suggest the following "best practices" when writing or using functions that deal with point clouds:

1. De-mean each cloud to center it at or close to the origin before operating on it. Some applications such as CloudCompare will explicitly ask the user to do this before loading a cloud with large coordinates.
2. Use or write functions which provide support for larger data types. For example, Libpointmatcher and some PCL functions are templated to allow the user to specify the data type as either float or double.
3. Explicitly check for conditions that indicate potential LoS (e.g., high DDR, large coordinates, and large numbers of significant digits in the result) and warn the user.
4. When dealing with LiDAR or other range data, it may be appropriate to store and process the cloud in the original reference frame of the sensor, and store its local or global coordinate offset separately.
5. In applications that use point normals, the authors recommend visualizing each point cloud and its normal vectors to confirm that they are orientated correctly. The `PCLVisualizer` API can be used for this purpose.

7. Conclusions

This paper has demonstrated that loss of significance (LoS) can be a large source of error for cloud registration and point normal calculation. The LoS in this context is a function of the cloud distance from origin and the mean point spacing.

When LoS occurs, it can result in highly random normal orientations. The degree of loss of significance, and the level of error that it generates varies depending on the cloud and the data type used. For a cloud with regular point spacing, such as the Stanford Bunny, the distance of any point from the origin must be 300–400 times the mean point spacing before 1° of error will occur in the point normal orientation. For irregular clouds such as those from a 3D LiDAR unit, this ratio can be as low as 60.

When LoS affects cloud registration, it can result in a sub-optimal match that is potentially several meters or degrees off from the ideal match. It can even cause a sub-component of the algorithm, such as correspondence detection, to fail completely. The Stanford Bunny results show that, when this happens, it can cause the algorithm as a whole to fail.

The experimental results provided show that the problem can be largely avoided by processing point clouds while they are close to or at the origin of the local coordinate system. Several best-practice recommendations have also been proposed, which readers may find useful for avoiding this problem.

Author Contributions: Conceptualization, M.Y.; formal analysis, M.Y., S.A., and C.P.; investigation, M.Y. and S.A.; methodology, M.Y., S.A., and C.P.; software, M.Y. and S.A.; supervision, C.P., R.G., and X.C.; visualization, M.Y.; writing—original draft, M.Y.; and writing—review and editing, M.Y., C.P., S.A., R.G., and X.C.

Funding: This research was jointly funded by Trimble Navigation New Zealand and Callaghan Innovation under research contract TNNX1601.

Acknowledgments: The authors would like to thank Luke Johnson for the SX10 dataset, as well as Josh McCulloch and Sergey Alexandrov for their expertise and help in identifying how loss of significance occurs within PCL.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rusu, R.B.; Cousins, S. 3D is here: Point Cloud Library (PCL). In Proceedings of the International Conference on Robotics and Automation (ICRA), Shanghai, China, 9–13 May 2011; pp. 1–4.
2. Klasing, K.; Althoff, D.; Wollherr, D.; Buss, M. Comparison of Surface Normal Estimation Methods for Range Sensing Applications. In Proceedings of the International Conference on Robotics and Automation, Kobe, Japan, 12–17 May 2009; pp. 3206–3211.
3. Jordan, K.; Mordohai, P. A Quantitative Evaluation of Surface Normal Estimation in Point Clouds. In Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robotics and Systems, Chicago, IL, USA, 14–18 September 2014; pp. 4220–4226.
4. Jin, S.; Lewis, R.R.; West, D. A Comparison of of Algorithms for Vertex Normal Computation. *Vis. Comput.* **2005**, *21*, 71–82. [CrossRef]
5. Hoppe, H.; DeRose, T.; Duchamp, T.; McDonald, J.; Stuetzle, W. Surface Reconstruction from Unorganized Points. In Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques, Chicago, IL, USA, 27–31 July 1992; pp. 71–78.
6. Gumhold, S.; Wang, X.; Macleod, R. Feature Extraction from Point Clouds. In Proceedings of the 10th International Meshing Roundtable, Newport Beach, CA, USA, 7–10 October 2001; pp. 293–305.
7. Tombari, F.; Salti, S.; Di Stefano, L. Unique Signatures of Histograms for Local Surface Description. In *Computer Vision—ECCV 2010, Proceedings of the European Conference on Computer Vision, Heraklion, Greece, 5–11 September 2010*; Springer: Berlin, Germany, 2010; pp. 356–369.
8. Mitra, N.J.; Nguyen, A. Estimating Surface Normals in Noisy Cloud Data. In Proceedings of the 19th Annual Symposium on Computational Geometry, San Diego, CA, USA, 8–10 June 2003; pp. 322–328.
9. Hoffman, R.; Jain, A.K. Segmentation and Classification of Range Images. *IEEE Trans. Pattern Anal. Mach. Intell.* **1987**, *9*, 608–620. [CrossRef] [PubMed]
10. Wang, C.; Tanahashi, H.; Hirya, H.; Niwa, Y.; Yamamoto, K. Comparison of Local Plane Fitting Methods for Range Data. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Kauai, HI, USA, 8–14 December 2001; pp. 663–669.
11. Besl, P.J.; McKay, N.D. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* **1992**, *14*, 239–256. [CrossRef]

12. Chen, Y.; Medioni, G. Object Modelling by Registration of Multiple Range Images. In Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Nice, France, 9–11 April 1991; pp. 2724–2729.
13. Segal, A.V.; Hahnel, D.; Thrun, S. Generalized ICP. *Robot. Sci. Syst.* **2009**, *2*, 435.
14. Biber, P.; Straßer, W. The Normal Distribution Transform: A New Approach to Laser Scan Matching. In Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, Las Vegas, NV, USA, 27–31 October 2003; pp. 2743–2748.
15. Rusinkiewicz, S.; Leroy, M. Efficient variants of the ICP algorithm. In Proceedings of the Third International Conference on 3-D Digital Imaging and Modelling, Quebec City, QC, Canada, 28 May–1 June 2001; pp. 145–152.
16. Pomerleau, F.; Colas, F.; Siegwart, R.; Magnenat, S. Comparing ICP Variants on Real-World Datasets. *Auton. Robots* **2013**, *34*, 133–148. [CrossRef]
17. Donoso, F.A.; Austin, K.J.; McAree, P.R. How do ICP variants perform when used for scan matching terrain point clouds? *Robot. Auton. Syst.* **2017**, *87*, 147–161. [CrossRef]
18. IEEE. *IEEE Standard for Floating-Point Arithmetic*; IEEE Std 754-2008; IEEE: Piscataway, NJ, USA, 2008.
19. Rajaraman, V. IEEE Standard for Floating Point Numbers. *Resonance* **2016**, *21*, 11–30. [CrossRef]
20. Goldberg, D. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Comput. Surv.* **1991**, *23*, 5–48. [CrossRef]
21. Low, K.L. *Linear Least-Squares Optimizations for Point-to-Plane ICP Surface Registration*; Technical Report TR04-004; Department of Computer Science, University of North Carolina: Chapel Hill, NC, USA, 2004.